DEFINIÇÃO

Consultas com expressões no comando SELECT. Consultas com o uso da cláusula WHERE. Agrupamento de dados.

PROPÓSITO

Saber construir comandos SQL com o uso de expressões no comando SELECT, bem como a especificação de condições na cláusula WHERE, que representam tarefas importantes no projeto de consultas em sistemas gerenciadores de banco de dados (SGBD). Para o desenvolvimento de relatórios e consultas analíticas, é fundamental saber trabalhar com agrupamento de dados. Essas atividades são relacionadas ao dia a dia de programadores, analistas de sistemas e desenvolvedores.

PREPARAÇÃO

Antes de iniciar o conteúdo deste tema, certifique-se de ter baixado e instalado o SGBD PostgreSQL em seu computador.

OBJETIVOS

MÓDULO 1

Operar consultas com o comando SELECT

MÓDULO 2

Operar consultas usando a cláusula WHERE

MÓDULO 3

Operar consultas envolvendo agrupamento de dados

INTRODUÇÃO

Ao longo deste tema, vamos explorar diversos exemplos de consultas envolvendo uma tabela. Aprenderemos a codificar consultas abrangendo tanto a recuperação de colunas da própria tabela quanto o uso de expressões no comando SELECT. Quando projetamos um banco de dados para determinado domínio de negócio, em geral, são criadas diversas tabelas que serão manipuladas pelas aplicações desenvolvidas para acessar os recursos do banco de dados.

Diversas operações que manipulam tabelas em um banco de dados necessariamente estão associadas a alguma operação de consulta. Por exemplo, se resolvermos aumentar em 10% o

salário de todos os funcionários que ganham até R$ 4.000, será necessário programarmos um comando de consulta para que o sistema gerenciador de banco de dados (SGBD) selecione os registros dos funcionários alvo da atualização. Assim, aprender de maneira efetiva a programar consultas trará benefícios, tanto para atividades de construção de relatórios, quanto para o projeto de operações de remoção e atualização de dados.

Clique aqui para baixar o arquivo com todos os códigos que serão utilizados nas consultas dos módulos a seguir.

MÓDULO 1

- **Operar consultas com o comando SELECT**

ESTRUTURA BÁSICA DE UM COMANDO SELECT

O comando SELECT é usado para exibir dados resultantes de uma consulta. Os dados podem ser colunas físicas de uma tabela, colunas calculadas ou mesmo resultado do uso de expressões e funções. Uma sintaxe **básica** para o comando SELECT está expressa a seguir:

```
SELECT COLUNA1 [[AS] APELIDOCOLUNA1],
COLUNA2 [[AS] APELIDOCOLUNA2],
…
COLUNAN [[AS] APELIDOCOLUNAN]
FROM TABELA;
```

É importante ressaltar que estamos diante de uma sintaxe simplificada o suficiente para entendimento dos exemplos que iremos explorar ao longo do módulo. A sintaxe completa abrange todos os recursos do PostgreSQL.

Uma sintaxe complexa envolve uma série de cláusulas e recursos bastante úteis para consultas de maior complexidade.

Na prática, o comando SELECT, dependendo da consulta desejada, pode ser usado de diferentes formas para obter o mesmo resultado. É importante frisar que a cláusula SELECT realiza a operação de projeção da Álgebra Relacional.

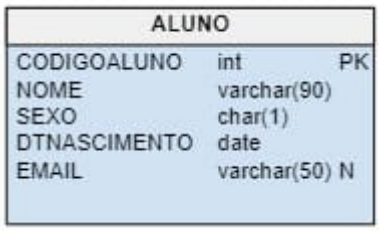
Caso haja interesse em exibir todas as colunas especificadas em uma consulta, basta adicionar um “*”, conforme a seguir: SELECT * FROM TABELA;

❓ VOCÊ SABIA

Alguns SGBDs, como o PostgreSQL, implementam uma forma simplificada do comando SELECT * FROM TABELA, que é simplesmente TABLE tabela (você pode testar isso no PostgreSQL).

Vamos estudar alguns exemplos?

Construiremos as consultas com base na tabela ALUNO, conforme figura a seguir:



Fonte: O autor
Tabela ALUNO.

Recomendamos que você crie a tabela e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência.

Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessando algum *database* criado por você.

```
1  CREATE TABLE ALUNO (
2     CODIGOALUNO int  NOT NULL,
3     NOME varchar(90)  NOT NULL,
4     SEXO char(1)  NOT NULL,
5     DTNASCIMENTO date  NOT NULL,
6     EMAIL varchar(30)  NULL,
7     CONSTRAINT ALUNO_pk PRIMARY KEY (CODIGOALUNO));
8
9  INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (1,'JOSÉ FRANCISCO TERRA','M','28/10/1989','JFT@GMAIL.COM');
10 INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (2,'ANDREY COSTA FILHO','M','20/10/1999','ANDREYCF@HOTMAIL.COM');
11 INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (3,'PATRÍCIA TORRES LOUREIRO','F','20/10/1980', 'PTORRES@GMAIL.COM');
12 INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (4,'CARLA MARIA MACIEL','F','20/11/1996',NULL);
13 INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (5,'LEILA SANTANA COSTA','F','20/11/2001',NULL);
```

Fonte: O autor.

Vamos ver alguns exemplos de consultas?

CONSULTA 01

Exibir todas as informações dos alunos.

SELECT * FROM ALUNO;

A tabela a seguir apresenta os resultados da consulta.

	codigoaluno	nome	sexo	dtnascimento	email
1	1	JOSÉ FRANCISCO TERRA	M	1989-10-28	JFT@GMAIL.COM
2	2	ANDREY COSTA FILHO	M	1999-10-20	ANDREYCF@HOTMAIL.COM
3	3	PATRÍCIA TORRES LOUREIRO	F	1980-10-20	PTORRES@GMAIL.COM
4	4	CARLA MARIA MACIEL	F	1996-11-20	[NULL]
5	5	LEILA SANTANA COSTA	F	2001-11-20	[NULL]

Fonte: O autor.

📷 Resultados da consulta 01.

Ao executar a consulta, o SGBD percorre todos os registros da tabela ALUNO e exibe as colunas dessa tabela.

CONSULTA 02

Retornar o código, o nome e a data de nascimento de todos os alunos.

SELECT CODIGOALUNO, NOME, DTNASCIMENTO
FROM ALUNO;

	codigoaluno	nome	dtnascimento
1	1	JOSÉ FRANCISCO TERRA	1989-10-28
2	2	ANDREY COSTA FILHO	1999-10-20
3	3	PATRÍCIA TORRES LOUREIRO	1980-10-20
4	4	CARLA MARIA MACIEL	1996-11-20
5	5	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor.

📷 Resultados da consulta 02.

Na consulta 02, foram especificadas três colunas da tabela ALUNO para serem exibidas ao usuário.

Em especial, pode ser interessante “renomear” as colunas resultantes da consulta, visando tornar os resultados mais “apresentáveis” ao usuário da aplicação. Por exemplo, a consulta 02 pode ser reescrita conforme a seguir:

SELECT CODIGOALUNO AS "Matrícula",
NOME AS "Nome do discente",

DTNASCIMENTO AS "Data de nascimento"

FROM ALUNO;,

O resultado dessa consulta seria este:

	Matrícula	Nome do discente	Data de nascimento
1	1	JOSÉ FRANCISCO TERRA	1989-10-28
2	2	ANDREY COSTA FILHO	1999-10-20
3	3	PATRÍCIA TORRES LOUREIRO	1980-10-20
4	4	CARLA MARIA MACIEL	1996-11-20
5	5	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultados da segunda versão da consulta 02.

É importante ressaltar que, na tabela anterior, o nome apresentado para cada coluna não existe fisicamente no banco de dados.

Vamos aprender a seguir que nem toda coluna resultante de uma consulta representa necessariamente uma coluna de alguma tabela.

FUNÇÕES DE DATA E HORA

Quando desenvolvemos consultas, é comum manipularmos colunas e funções que envolvem dados representativos de datas.

Funções de data do PostgreSQL.

Um resumo contendo algumas funções de data do PostgreSQL pode ser visualizado na tabela a seguir:	
Função	**O que retorna?**
current_date	data de hoje
current_time	hora do dia
current_timestamp	data e a hora

extract (campo from fonte)	subcampos de data e hora: século, ano, dia, mês

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Fonte: O autor.

Um quadro completo contendo informações sobre funções de data e hora pode ser encontrado na documentação oficial do PostgreSQL.

Vamos estudar alguns exemplos?

Observe com atenção o código:

```
SELECT CURRENT_DATE AS "Data Atual",
       CURRENT_TIME AS "Hora Atual",
       CURRENT_TIMESTAMP "Data e Hora atuais",
       EXTRACT( DOY FROM CURRENT_DATE) AS "Dia do ano",
-- DOW 0 - domingo, 1 - segunda, ..., 6 - sábado
       EXTRACT( DOW FROM CURRENT_DATE) AS "Dia da semana",
       EXTRACT( DAY FROM CURRENT_DATE) AS "Dia Atual",
       EXTRACT( MONTH FROM CURRENT_DATE) AS "Mês Atual",
       EXTRACT( YEAR FROM CURRENT_DATE) AS "Ano Atual",
       EXTRACT( CENTURY FROM CURRENT_DATE) AS "Século Atual";
```

Fonte: O autor

Agora veja na tabela a seguir os resultados da consulta:

	Data Atual	Hora Atual	Data e Hora atuais	Dia do ano	Dia da semana	Dia Atual	Mês Atual	Ano Atual	Século Atual
1	2020-06-24	08:55:13	2020-06-24 08:55:13	176	3	24	6	2.020	21

Fonte: O autor.

📷 Resultados da consulta envolvendo funções de data e hora.

Observe que utilizamos o qualificador *AS "Apelido"* para facilitar o entendimento do retorno de cada função. Note também que não há cláusula FROM na consulta, visto que todas as colunas retornadas representam o resultado de funções do PostgreSQL sem envolver qualquer tabela do domínio da aplicação.

📢 ATENÇÃO

Convém ressaltar que, no padrão SQL, a cláusula FROM é obrigatória. No entanto, o PostgreSQL permite executar um comando SELECT **sem** a cláusula FROM. Experimente executar SELECT 5+5;

EXIBINDO O NOME DO DIA DA SEMANA

Perceba que a linha 6 do código acima retorna um inteiro representativo do dia da semana. No entanto, se houver necessidade de exibir o dia da semana, você pode usar o código a seguir:

```
1 SELECT CASE WHEN extract(dow from current_date) = 0 THEN 'domingo'
2             WHEN extract(dow from current_date) = 1 THEN 'segunda-feira'
3             WHEN extract(dow from current_date) = 2 THEN 'terça-feira'
4             WHEN extract(dow from current_date) = 3 THEN 'quarta-feira'
5             WHEN extract(dow from current_date) = 4 THEN 'quinta-feira'
6             WHEN extract(dow from current_date) = 5 THEN 'sexta-feira'
7             WHEN extract(dow from current_date)= 6 THEN 'sábado'
8        END AS "Nome do dia da semana";
```

Fonte: O autor

Observe que construímos uma lógica utilizando o comando CASE, que é equivalente ao comando IF:, cada linha com a cláusula WHEN avalia expressão que retorna um inteiro representativo do dia da semana, caso a expressão tenha valor lógico verdadeiro.

CALCULANDO IDADE E FAIXA ETÁRIA

Em geral, quando estamos diante de alguma coluna representativa da data de nascimento de uma pessoa, é comum extrair informações derivadas, tais como idade e faixa etária. Por exemplo, o código a seguir retorna o nome, a data de nascimento e a idade dos alunos:

```
1 SELECT  NOME,
2         DTNASCIMENTO,
3         AGE(DTNASCIMENTO) AS "Idade [ano/mês/dia]",
4         EXTRACT(YEAR FROM AGE(DTNASCIMENTO))    AS "Idade do Aluno"
5 FROM ALUNO;
```

Fonte: O autor

Perceba que na linha 3 utilizamos a função AGE, que retorna uma frase representativa da informação sobre a idade em questão. Na linha 4, usamos a função EXTRACT para exibir a

idade do aluno. A figura a seguir apresenta o resultado dessa consulta:

	nome	dtnascimento	Idade [ano/mês/dia]	Idade do Aluno
1	JOSÉ FRANCISCO TERRA	1989-10-28	30 years 7 mons 30 days	30
2	ANDREY COSTA FILHO	1999-10-20	20 years 8 mons 7 days	20
3	PATRÍCIA TORRES LOUREIRO	1980-10-20	39 years 8 mons 7 days	39
4	CARLA MARIA MACIEL	1996-11-20	23 years 7 mons 7 days	23
5	LEILA SANTANA COSTA	2001-11-20	18 years 7 mons 7 days	18

Fonte: O autor

📷 Exibindo a idade dos alunos.

Muito bem, agora, vamos exibir o nome, a idade e a faixa etária dos alunos.

Observe o código SQL a seguir:

```sql
SELECT  NOME,
        EXTRACT(YEAR FROM AGE(DTNASCIMENTO))    AS "Idade do Aluno",
        CASE WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) <=20  THEN '1. até 20 anos'
             WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 21 AND 30  THEN '2. 21 a 30 anos'
             WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 31 AND 40  THEN '3. 31 a 40 anos'
             WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 41 AND 50  THEN '4. 41 a 50 anos'
             WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 51 AND 60  THEN '5. 51 a 60 anos'
             WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) > 60  THEN '6. mais de 60 anos'
        END AS "Faixa Etária"
FROM ALUNO;
```

Fonte: O autor

Perceba que cada linha com a cláusula WHEN avalia a expressão que retorna uma faixa etária de acordo com a idade do aluno.

A seguir, o resultado da consulta:

	nome	Idade do Aluno	Faixa Etária
1	JOSÉ FRANCISCO TERRA	30	2. 21 a 30 anos
2	ANDREY COSTA FILHO	20	1. até 20 anos
3	PATRÍCIA TORRES LOUREIRO	39	3. 31 a 40 anos
4	CARLA MARIA MACIEL	23	2. 21 a 30 anos
5	LEILA SANTANA COSTA	18	1. até 20 anos

Fonte: O autor.

📷 Resultados da consulta envolvendo idade e faixa etária dos alunos.

FUNÇÕES DE RESUMO OU DE AGREGAÇÃO

As funções a seguir são úteis para obtermos resumo dos dados de alguma tabela:

Funções para resumo de dados.

Função	**O que retorna?**
COUNT(*)	número de linhas da consulta
MIN(COLUNA/EXPRESSÃO)	menor de uma coluna ou expressão
AVG(COLUNA/EXPRESSÃO)	valor médio da coluna ou expressão
MAX(COLUNA/EXPRESSÃO)	maior valor de uma coluna ou expressão
SUM(COLUNA/EXPRESSÃO)	soma dos valores de uma coluna ou expressão
STDDEV(COLUNA/EXPRESSÃO)	desvio padrão dos valores de uma coluna ou expressão
VARIANCE(COLUNA/EXPRESSÃO)	variância dos valores de uma coluna ou expressão

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Fonte: O Autor.

Vamos estudar um exemplo?

Observe o código a seguir:

```
1 SELECT
2         COUNT(*) AS "Número de alunos",
3         MIN(EXTRACT(YEAR FROM AGE(DTNASCIMENTO)))   AS "Menor Idade",
4         AVG(EXTRACT(YEAR FROM AGE(DTNASCIMENTO)))   AS "Idade Média",
5         MAX(EXTRACT(YEAR FROM AGE(DTNASCIMENTO)))   AS "Maior Idade",
6         SUM(EXTRACT(YEAR FROM AGE(DTNASCIMENTO)))/COUNT(*) AS "Idade Média"
7 FROM ALUNO;
```

Fonte: O autor

Perceba que, como estamos usando somente o comando SELECT/FROM, cada função é calculada levando em consideração todos os registros da tabela.

Veja na figura a seguir, o resultado da consulta:

	Número de alunos	Menor Idade	Idade Média	Maior Idade	Idade Média
1	5	18	26	39	26

Fonte: O autor.

📷 Resultado da consulta envolvendo funções para extrair resumo a partir da tabela aluno.

📢 ATENÇÃO

Perceba, também, que o código da linha 6 é equivalente ao da linha 4: ambos calculam a idade média dos alunos.

LISTANDO RESUMOS EM UMA LINHA

Suponha que haja interesse em conhecer os quantitativos de cursos, disciplinas e alunos do nosso banco de dados.

Poderíamos submeter ao SGBD as consultas a seguir:

CURSO

SELECT COUNT(*) NCURSOS FROM CURSO;

DISCIPLINA

SELECT COUNT(*) NDISCIPINAS FROM DISCIPLINA;

ALUNO

SELECT COUNT(*) NALUNOS FROM ALUNO;

Estamos diante de três consultas. No entanto, pode ser mais interessante mostrarmos os resultados em apenas uma linha.

Podemos, então, submeter o código a seguir:

```
1 SELECT
2  (SELECT COUNT(*) NCURSOS FROM CURSO),
3  (SELECT COUNT(*) NALUNOS FROM ALUNO),
4  (SELECT COUNT(*) NDISCIPINAS FROM DISCIPLINA);
```

Fonte: O autor

O que fizemos?

Como cada consulta (linhas 2 a 4) retorna somente um valor, utilizamos um SELECT externo (linha 1) para exibir cada coluna resultante.

Observe o resultado a seguir:

	ncursos	nalunos	ndiscipinas
1	4	5	4

Fonte: O autor.

📷 Resultado da consulta envolvendo quantitativos de cursos, alunos e disciplinas.

Convém ressaltar que o comando é válido, visto que, no PostgreSQL, a cláusula FROM não é obrigatória.

CRIANDO TABELA A PARTIR DE CONSULTA

Em alguns momentos, você terá interesse em salvar os resultados de uma consulta em uma nova tabela.

Para isso, basta usar o comando CREATE TABLE <CONSULTA>.

```
1  CREATE TABLE TTESTE AS
2  SELECT  NOME,
3          EXTRACT(YEAR FROM AGE(DTNASCIMENTO))    AS "Idade do Aluno",
4          CASE WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) <=20  THEN '1. até 20 anos'
5               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 21 AND 30  THEN '2. 21 a 30 anos'
6               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 31 AND 40  THEN '3. 31 a 40 anos'
7               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 41 AND 50  THEN '4. 41 a 50 anos'
8               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 51 AND 60  THEN '5. 51 a 60 anos'
9               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) > 60  THEN '6. mais de 60 anos'
10         END AS "Faixa Etária"
11 FROM ALUNO;
```

Fonte: O autor

No exemplo apresentado, o SGBD criará uma tabela denominada TTESTE e armazenará os dados resultantes da consulta (linhas 2 a 11) em questão.

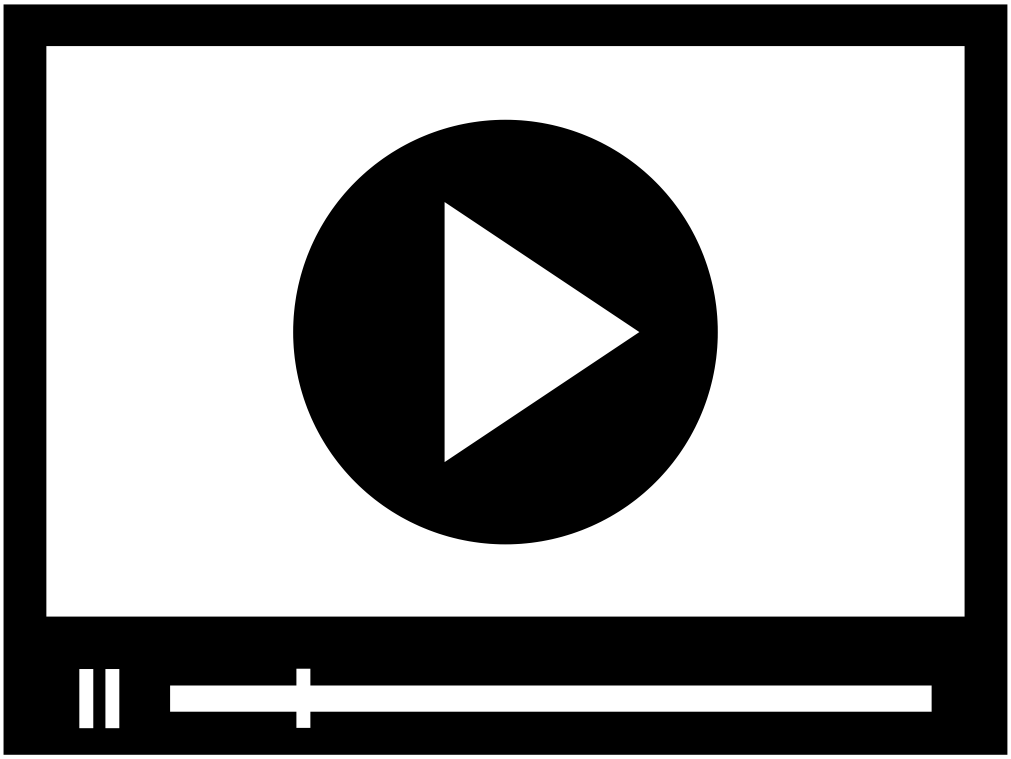
CRIANDO *VIEW* A PARTIR DE CONSULTA

Outro recurso interessante, diretamente relacionado ao processo de construção de consultas, é o objeto *view* (visão). Uma *view* encapsula a complexidade da consulta SQL, que a forma. Para criar esse objeto, usa-se o comando CREATE VIEW <CONSULTA>.

```
1  CREATE VIEW VTESTE AS
2  SELECT  NOME,
3          EXTRACT(YEAR FROM AGE(DTNASCIMENTO))    AS "Idade do Aluno",
4          CASE WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) <=20  THEN '1. até 20 anos'
5               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 21 AND 30  THEN '2. 21 a 30 anos'
6               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 31 AND 40  THEN '3. 31 a 40 anos'
7               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 41 AND 50  THEN '4. 41 a 50 anos'
8               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) BETWEEN 51 AND 60  THEN '5. 51 a 60 anos'
9               WHEN EXTRACT(YEAR FROM AGE(DTNASCIMENTO)) > 60  THEN '6. mais de 60 anos'
10         END AS "Faixa Etária"
11 FROM ALUNO;
```

Fonte: O autor

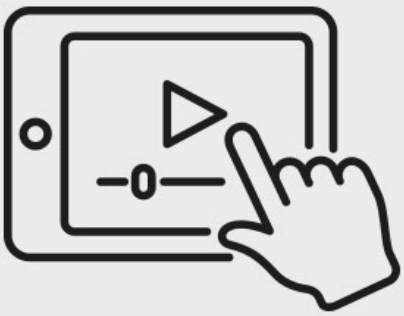
No exemplo, o SGBD criará uma *view* denominada VTESTE. Na prática, quando usuário submeter, por exemplo, a consulta SELECT * FROM VTESTE, o SGBD executará o código associado à *view* em questão.



CONSULTAS SIMPLES COM O COMANDO SELECT NO POSTGRESQL

Veja agora o vídeo sobre **Consultas simples com o comando SELECT no PostgreSQL**

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Ao longo da nossa jornada, estudamos a construção de consultas envolvendo a extração de informação a partir de uma tabela. Além disso, foram exibidas funções de data e funções para resumir dados de uma tabela.

Agora é com você! Vamos realizar as atividades a seguir?

VERIFICANDO O APRENDIZADO

1. CONSIDERE A TABELA E O CÓDIGO SQL A SEGUIR:

ALUNO		
CODIGOALUNO	int	PK
NOME	varchar(90)	
SEXO	char(1)	
DTNASCIMENTO	date	

```
1 SELECT  CURRENT_DATE AS "Data de Emissão do Relatório",
2         CODIGOALUNO AS "Matrícula",
3         NOME AS "Nome do discente",
4         DTNASCIMENTO AS "Data de nascimento",
5         CASE
6             WHEN  SEXO='M' THEN 'Masculino'
7             WHEN  SEXO='F' THEN 'Feminino'
8         END AS SEXO
9 FROM ALUNO;
```

ANALISE AS SEGUINTES PROPOSIÇÕES:

I - A CONSULTA RETORNA INFORMAÇÕES SOBRE CINCO COLUNAS EXISTENTES NA TABELA ALUNO.
II - A CONSULTA RETORNA INFORMAÇÕES SOBRE TODOS OS ALUNOS CADASTRADOS.
III - PODE EXISTIR REGISTRO COM VALOR “MASCULINO” ARMAZENADO NA COLUNA SEXO.
IV - O RESULTADO DE CURRENT_DATE (LINHA 1) ESTÁ ARMAZENADO EM UMA COLUNA DA TABELA ALUNO.
V - A CONSULTA RETORNA INFORMAÇÕES SOBRE QUATRO COLUNAS EXISTENTES NA TABELA ALUNO.

SÃO PROPOSIÇÕES VERDADEIRAS:

A) I e II.

B) II e V.

C) II, III e IV.

D) III e V.

2. SEJA UMA TABELA ASSIM DEFINIDA: FUNCIONARIO (IDFUNC, NOME, DATANASCIMENTO, SALARIO). QUAL CONSULTA SQL RETORNA O MAIOR SALÁRIO?

A) SELECT > SALARIO FROM FUNCIONARIO.

B) SELECT MAX(SALARIO) FROM FUNCIONARIO.

C) SELECT AVG(SALARIO) FROM FUNCIONARIO.

D) SELECT FUNCIONARIO FROM SALÁRIO.

GABARITO

1. Considere a tabela e o código SQL a seguir:

ALUNO		
CODIGOALUNO	int	PK
NOME	varchar(90)	
SEXO	char(1)	
DTNASCIMENTO	date	

```sql
SELECT  CURRENT_DATE AS "Data de Emissão do Relatório",
        CODIGOALUNO AS "Matrícula",
        NOME AS "Nome do discente",
        DTNASCIMENTO AS "Data de nascimento",
        CASE
            WHEN  SEXO='M' THEN 'Masculino'
            WHEN  SEXO='F' THEN 'Feminino'
        END AS SEXO
FROM ALUNO;
```

Analise as seguintes proposições:

I - A consulta retorna informações sobre cinco colunas existentes na tabela ALUNO.

II - A consulta retorna informações sobre todos os alunos cadastrados.

III - Pode existir registro com valor “Masculino” armazenado na coluna SEXO.

IV - O resultado de CURRENT_DATE (linha 1) está armazenado em uma coluna da tabela ALUNO.

V - A consulta retorna informações sobre quatro colunas existentes na tabela ALUNO.

São proposições verdadeiras:

A alternativa **"B "** está correta.

A proposição II é verdadeira, pois não há condição de filtro na consulta. A proposição V é verdadeira, pois retorna informações a respeito de todas as colunas da tabela ALUNO. As demais proposições são falsas.

2. Seja uma tabela assim definida: FUNCIONARIO (IDFUNC, NOME, DATANASCIMENTO, SALARIO). Qual consulta SQL retorna o maior salário?

A alternativa **"B "** está correta.

Na alternativa B, foi usado o comando MAX para retornar o maior valor da coluna SALÁRIO da tabela FUNCIONARIO.

MÓDULO 2

⦿ **Operar consultas usando a cláusula WHERE**

CLÁUSULA WHERE E OPERADORES DA SQL

Em nossas consultas, usaremos como base a tabela ALUNO, conforme figura a seguir:

ALUNO		
CODIGOALUNO	int	PK
NOME	varchar(90)	
SEXO	char(1)	
DTNASCIMENTO	date	
EMAIL	varchar(50)	N

Fonte: O autor

📷 Tabela ALUNO.

Recomendamos que você crie a tabela e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessando algum *database* criado por você.

```sql
CREATE TABLE ALUNO (
   CODIGOALUNO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   SEXO char(1)  NOT NULL,
   DTNASCIMENTO date  NOT NULL,
   EMAIL varchar(30)  NULL,
   CONSTRAINT ALUNO_pk PRIMARY KEY (CODIGOALUNO));

INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (1,'JOSÉ FRANCISCO TERRA','M','28/10/1989','JFT@GMAIL.COM');
INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (2,'ANDREY COSTA FILHO','M','20/10/1999','ANDREYCF@HOTMAIL.COM');
INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (3,'PATRÍCIA TORRES LOUREIRO','F','20/10/1980', 'PTORRES@GMAIL.COM');
INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (4,'CARLA MARIA MACIEL','F','20/11/1996',NULL);
INSERT INTO ALUNO (CODIGOALUNO, NOME, SEXO, DTNASCIMENTO,EMAIL) VALUES   (5,'LEILA SANTANA COSTA','F','20/11/2001',NULL);
```

Fonte: O autor

RECUPERANDO DADOS COM SELECT/FROM/WHERE/ORDER BY

Uma sintaxe básica para o comando SELECT, com o uso das cláusulas WHERE e ORDER BY, está expressa a seguir:

SELECT COLUNA1 [[AS] APELIDOCOLUNA1],

COLUNA2 [[AS] APELIDOCOLUNA2],

…

COLUNAN [[AS] APELIDOCOLUNAN]

FROM TABELA

WHERE <CONDIÇÃO>

ORDER BY EXPRESSÃO1[ASC|DESC] [NULLS {FIRST|LAST}], [EXPRESSÃO2[ASC|DESC] [NULLS {FIRST|LAST}…];

O propósito do SELECT é declararmos as colunas da consulta. No FROM, informamos a tabela alvo da consulta. No WHERE, especificamos alguma condição, simples ou composta, para filtrar registros que serão recuperados pelo SGBD. No ORDER BY, declaramos uma ou mais colunas como critério de ordenação, com possibilidade de especificarmos se valores NULL aparecem no início ou no final do resultado.

É importante frisar que a cláusula WHERE realiza a operação de restrição da Álgebra Relacional, também conhecida como seleção – não confundir com o comando SELECT.

Ainda, a construção de uma condição na cláusula WHERE envolve operadores relacionais, conforme tabela a seguir:

Operadores relacionais.

Operador	Significado
<	menor
<=	menor ou igual a
>	maior
>=	maior ou igual a
=	igual
<> ou !=	> diferente

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Fonte: O autor.

Além dos operadores relacionais, a construção de uma condição na cláusula WHERE pode fazer uso dos seguintes operadores lógicos:

Operadores lógicos.

Operador	Significado
AND	conjunção
OR	disjunção
NOT	negação

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Fonte: O autor.

Vamos estudar alguns exemplos de consultas com o uso da cláusula WHERE?

CONSULTA 01 + RESULTADO

Mostrar o nome e a data de nascimento das professoras.

	nome	dtnascimento
1	PATRÍCIA TORRES LOUREIRO	1980-10-20
2	CARLA MARIA MACIEL	1996-11-20
3	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	PATRÍCIA TORRES LOUREIRO	1980-10-20
2	CARLA MARIA MACIEL	1996-11-20
3	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 01.

Perceba que foi criada uma condição simples de igualdade envolvendo a coluna SEXO da tabela ALUNO. O SGBD percorre cada registro da tabela ALUNO, avalia a condição (linha 3) e exibe as colunas NOME e DTNASCIMENTO para cada registro cuja avaliação da condição retorne verdadeiro.

CONSULTA 02 + RESULTADO

Mostrar o nome e a data de nascimento das professoras que fazem aniversário em novembro.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 02.

Perceba que foi criada uma condição composta envolvendo uma conjunção. O SGBD retornará os registros que possuem o valor “F” para a coluna SEXO e o inteiro 11 como valor do mês referente à data de nascimento.

RECUPERANDO DADOS COM O USO DO OPERADOR IN

O operador [NOT] IN pode ser utilizado em consultas que envolvam comparações usando uma lista de valores.

CONSULTA 03 + RESULTADO

Listar o nome dos alunos que fazem aniversário no segundo semestre.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 03.

Note que a expressão na cláusula WHERE compara o mês de nascimento de cada aluno junto aos valores da lista contendo os inteiros correspondentes aos meses do segundo semestre.

RECUPERANDO DADOS COM O USO DO OPERADOR BETWEEN

O operador [NOT] BETWEEN verifica se determinado valor encontra-se no intervalo entre dois valores.

Por exemplo, *X BETWEEN Y AND Z* é equivalente a *X>=Y AND X<=Z*. De modo semelhante, *X NOT BETWEEN Y AND Z* é equivalente a *X<Y OR X>Z*.

CONSULTA 04 + RESULTADO

Listar o nome dos alunos nascidos entre 1985 e 2005.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 04.

Note que a expressão na cláusula WHERE compara o ano de nascimento de cada aluno junto ao intervalo especificado pelo operador BETWEEN. Caso quiséssemos extrair o mesmo resultado sem o uso do BETWEEN, poderíamos programar um comando equivalente, conforme a seguir:

```sql
SELECT NOME
FROM ALUNO
WHERE  EXTRACT (YEAR FROM DTNASCIMENTO) >= 1985 AND EXTRACT (YEAR FROM DTNASCIMENTO) <= 2005;
```

Fonte: O autor

RECUPERANDO DADOS COM O USO DO OPERADOR LIKE

O uso do [NOT] LIKE permite realizar buscas em uma cadeia de caracteres.

Trata-se de um recurso bastante utilizado em buscas textuais. Você pode utilizar os símbolos especiais a seguir:

_ (Underline) para ignorar qualquer caractere específico;

% (Percentual) para ignorar qualquer padrão.

Vamos estudar alguns exemplos?

CONSULTA 05 + RESULTADO

Listar o nome dos alunos que possuem a *string* COSTA em qualquer parte do nome.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 05.

SAIBA MAIS

O uso do padrão ‘%COSTA%’ significa que não importa o conteúdo localizado antes e depois da *string* “COSTA”.

CONSULTA 06 + RESULTADO

Listar o nome dos alunos que possuem a letra “A” na segunda posição do nome.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 06.

Note que, para especificar o “A” na segunda posição, o SGBD desprezará qualquer valor na primeira posição da *string*, não importando o que estiver localizado à direita do “A”.

CONSULTA 07 + RESULTADO

Listar o nome e a data de nascimento dos alunos que não possuem a *string* “MARIA” fazendo parte do nome.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 07.

Estamos diante de um caso semelhante ao da consulta 05.

No entanto, utilizamos o operador de negação para retornar os registros de interesse.

CONSULTA 08 + RESULTADO

Quantos alunos possuem conta de e-mail no gmail?

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

📷 Resultado consulta 08.

Note que, mais uma vez, estamos diante de um caso semelhante ao da consulta 05. Buscamos pela string “@GMAIL. ” em qualquer posição da coluna EMAIL.

RECUPERANDO DADOS COM O USO DO OPERADOR NULL

Quando uma coluna é opcional, significa que existe possibilidade de que algum registro não possua valor cadastrado para a coluna em questão. Nessa hipótese, há entendimento de que o valor da coluna é “desconhecido” ou “não aplicável”.

Para testar se uma coluna possui valor cadastrado, usa-se a expressão COLUNA IS NOT NULL.

Vamos estudar alguns exemplos?

CONSULTA 09 + RESULTADO

Listar o nome, a data de nascimento e o e-mail dos alunos que têm endereço eletrônico cadastrado.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 09.

O SGBD retorna os registros onde há algum conteúdo cadastrado na coluna EMAIL.

CONSULTA 10 + RESULTADO

Retornar o nome dos alunos sem e-mail cadastrado no banco de dados.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 10.

O SGBD retorna os registros sobre os quais não há valor cadastrado na coluna EMAIL.

RECUPERANDO DADOS USANDO ORDENAÇÃO DOS RESULTADOS

Para melhor organizar os resultados de uma consulta, nós podemos especificar critérios de ordenação. Vejamos alguns exemplos:

CONSULTA 11 + RESULTADO

Retornar o nome e a data de nascimento dos alunos, ordenando os resultados por nome, de maneira ascendente.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 11.

O SGBD retorna os registros da tabela ALUNO, obedecendo ao critério de ordenação especificado na linha 3 da consulta. O padrão ascendente (ASC) é opcional.

CONSULTA 12 + RESULTADO

Retornar o nome e a data de nascimento dos alunos, ordenando os resultados de modo ascendente pelo mês de nascimento e, em seguida, pelo nome, também de modo ascendente.

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

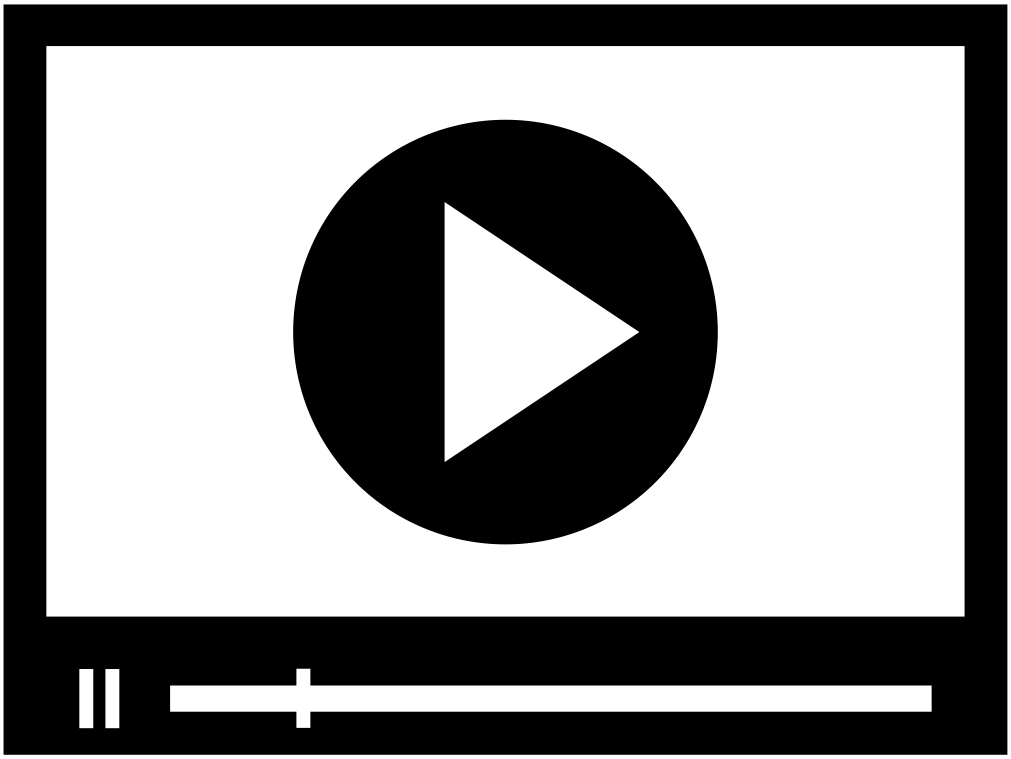
Fonte: O autor

	nome	dtnascimento
1	CARLA MARIA MACIEL	1996-11-20
2	LEILA SANTANA COSTA	2001-11-20

Fonte: O autor

Resultado consulta 12.

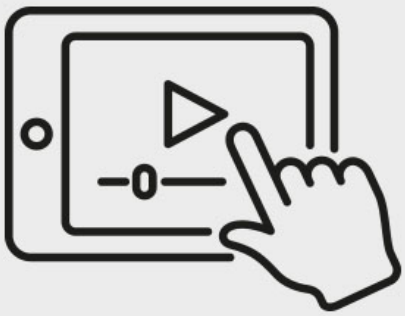
O SGBD retorna os registros da tabela ALUNO, levando em conta o critério de ordenação especificado na linha 3 da consulta. Foi realizada ordenação pelo mês de nascimento; em seguida, pelo nome.



CONSULTAS COM O COMANDO SELECT E A CLÁUSULA WHERE

A seguir, veja o vídeo: **Consultas com o comando SELECT e a cláusula WHERE**

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Trabalhamos o uso de consultas com auxílio da cláusula WHERE, fazendo uso de operadores relacionais e lógicos na composição de condições lógicas. Além disso, estudamos como estabelecer critérios para ordenação dos resultados de consultas.

Agora é com você! Vamos realizar as atividades a seguir?

VERIFICANDO O APRENDIZADO

1. GABRIEL É ANALISTA DE SISTEMAS DE UMA EMPRESA DE TECNOLOGIA DE INFORMAÇÃO E RECEBEU A TAREFA DE RECUPERAR TODOS OS REGISTROS DA TABELA CLIENTE ONDE O VALOR DA COLUNA “NOMECLIENTE” CONTENHA A CADEIA “DA SILVA” EM QUALQUER PARTE DO NOME. ASSINALE A ALTERNATIVA CORRETA QUE PERMITA A GABRIEL EXECUTAR SUA TAREFA.

A) SELECT * FROM CLIENTE WHERE NOMECLIENTE = 'da Silva '.

B) SELECT * FROM CLIENTE WHERE NOMECLIENTE != 'da Silva '.

C) SELECT * FROM CLIENTE WHERE NOME LIKE '%da Silva% '.

D) SELECT * FROM CLIENTE WHERE NOMECLIENTE LIKE '%da Silva% '.

2. UM PROGRAMADOR RECUPEROU OS DADOS DOS BAIRROS PENHA, IPANEMA, FLAMENGO E CENTRO GRAVADOS NA COLUNA BAIRRO DA TABELA CLIENTE, A SEGUIR ESPECIFICADA.
CLIENTE (IDCLIENTE, NOME, ENDERECO, BAIRRO, CIDADE, UF, CEP)
A SINTAXE SQL CORRETA USADA POR ELE PARA REALIZAR ESSA ATIVIDADE FOI SELECT * FROM CLIENTE

A) WHERE BAIRRO IN ('Penha ', 'Ipanema ', 'Flamengo ', 'Centro').

B) WHERE BAIRRO = ('Penha ', 'Ipanema ', 'Flamengo ', 'Centro').

C) WHEN BAIRRO = ('Penha ', 'Ipanema ', 'Flamengo ', 'Centro').

D) WHERE BAIRRO BETWEEN ('Penha ', 'Ipanema ', 'Flamengo ', 'Centro').

GABARITO

1. Gabriel é analista de sistemas de uma empresa de tecnologia de informação e recebeu a tarefa de recuperar todos os registros da tabela CLIENTE onde o valor da coluna

“NOMECLIENTE” contenha a cadeia “da Silva” em qualquer parte do nome. Assinale a alternativa correta que permita a Gabriel executar sua tarefa.

A alternativa **"D "** está correta.

Para recuperar os registros que contenham “da Silva” em qualquer parte do nome, utiliza-se o comando LIKE com auxílio do “%” como forma do SGBD desconsiderar qualquer padrão à esquerda e à direita da string de interesse.

2. Um programador recuperou os dados dos bairros Penha, Ipanema, Flamengo e Centro gravados na coluna BAIRRO da tabela CLIENTE, a seguir especificada.
CLIENTE (IDCLIENTE, NOME, ENDERECO, BAIRRO, CIDADE, UF, CEP)
A sintaxe SQL correta usada por ele para realizar essa atividade foi SELECT * FROM CLIENTE

A alternativa **"A "** está correta.

Para recuperar os registros de interesse, foi utilizado o operador IN com o uso de uma lista contendo os bairros em questão. O SGBD compara o bairro do cliente junto aos elementos especificados na lista de bairros em questão.

MÓDULO 3

⦿ **Operar consultas envolvendo agrupamento de dados**

CONSULTAS COM GROUP BY E HAVING

Em nossas consultas, usaremos como base a tabela FUNCIONARIO, conforme figura a seguir:

FUNCIONARIO		
CODIGOFUNCIONARIO	int	PK
NOME	varchar(90)	
CPF	char(15)	
SEXO	char(1)	
DTNASCIMENTO	date	
SALARIO	real	N

Fonte: O autor

Tabela FUNCIONÁRIO.

Recomendamos que você crie a tabela e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessando algum *database* criado por você.

```sql
CREATE TABLE FUNCIONARIO (
   CODIGOFUNCIONARIO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CPF char(15)  NULL,
   SEXO char(1)  NOT NULL,
   DTNASCIMENTO date  NOT NULL,
   SALARIO real   NULL,
   CONSTRAINT FUNCIONARIO_pk PRIMARY KEY (CODIGOFUNCIONARIO));

INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO) VALUES (1,'ROBERTA SILVA BRASIL','82998','F','20/02/1980',7000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO) VALUES (2,'MARIA SILVA BRASIL','9876','F','20/09/1988',9500);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO) VALUES (3,'GABRIELLA PEREIRA LIMA','32998','F','20/02/1990',6000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO) VALUES (4,'MARCOS PEREIRA BRASIL','9999','M','20/02/1999',6000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO) VALUES (5,'HEMERSON SILVA BRASIL','9111','M','20/12/1992',4000);
```

Fonte: O autor.

Após a criação da tabela e a inserção dos registros, podemos utilizar o código a seguir para exibir todo o seu conteúdo:

```sql
SELECT *
FROM FUNCIONARIO;
```

Fonte: O autor

O resultado da consulta será semelhante a este:

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7000.0
2	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9500.0
3	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	6000.0
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	6000.0
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4000.0

Fonte: O autor

Registros da tabela FUNCIONARIO.

GRUPO DE DADOS

Nas próximas seções, vamos aprender a projetar consultas com o uso de agrupamento de dados, com auxílio dos comandos GROUP BY e HAVING.

Vamos perceber que a maior parte dessas consultas está atrelada ao uso de alguma função de resumo, por exemplo, SUM, AVG, MIN e MAX, as quais representam, respectivamente, soma, média, mínimo e máximo.

Logo, essas consultas são úteis para quem tem interesse em construir relatórios e aplicações de natureza mais gerencial e analítica. Os valores de determinada coluna podem formar grupos sobre os quais podemos ter interesse em recuperar dados.

Por exemplo, se avaliarmos o resultado da consulta anterior, podemos naturalmente dividir os registros de acordo com o valor da coluna sexo. Teríamos, então, uma estrutura conforme a seguir:

M {4,5} {MARCOS PEREIRA BRASIL, HEMERSON SILVA BRASIL} {…}

F {1,2,3} {ROBERTA SILVA BRASIL, MARIA SILVA BRASIL, GABRIELLA PEREIRA LIMA} {…}

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

SAIBA MAIS

Todas as linhas com o mesmo valor para a coluna sexo formam um grupo. Representamos as linhas com mais de um valor com o uso de chaves para fins ilustrativos, dado que estamos diante de linhas dentro de colunas, uma representação que não existe no modelo relacional.

A estrutura anterior possui somente um grupo, o qual é formado pela coluna SEXO da tabela FUNCIONARIO. Como exibir esse grupo em SQL?

Uma solução é adicionar a cláusula DISTINCT ao comando SELECT, conforme a seguir:

```
1 SELECT DISTINCT SEXO
2 FROM FUNCIONARIO;
```

Fonte: O autor

O resultado da consulta anterior pode ser visualizado na figura a seguir:

	sexo
1	M
2	F

Fonte: O autor.

📷 Grupo de dados baseado na coluna SEXO da tabela FUNCIONARIO.

Vamos perceber que em SQL a cláusula mais adequada para trabalhar com agrupamento de dados é o GROUP BY.

GRUPO DE DADOS COM GROUP BY

A cláusula GROUP BY serve para exibir resultados de consulta de acordo com um grupo especificado. Ela é declarada após a cláusula FROM, ou após a cláusula WHERE, caso exista na consulta. Por exemplo, para obter o mesmo resultado do comando anterior, podemos usar o código a seguir:

```
1 SELECT SEXO
2 FROM FUNCIONARIO
3 GROUP BY SEXO;
```

Fonte: O autor

No entanto, vamos perceber que o uso mais conhecido da cláusula GROUP BY ocorre quando associada a funções de agregação, tais como COUNT, MIN, MAX e AVG.

Uma tabela com o nome e o significado dessas funções foi apresentada na seção “**Funções de resumo ou agregação**” no módulo 1.

Vamos estudar alguns exemplos?

CONSULTA 01 + RESULTADO

Retornar o número de funcionários por sexo.

Fonte: O autor

Fonte: O autor

📷 Resultado consulta 01.

O SGBD realiza o agrupamento de dados de acordo com os valores da coluna SEXO. Em seguida, para cada grupo encontrado, a função COUNT(*) é executada e o resultado exibido.

E se tivéssemos interesse em exibir os resultados da consulta anterior em uma única linha? Poderíamos usar o código a seguir:

```
1 SELECT
2         (SELECT COUNT(*) AS "M" FROM FUNCIONARIO WHERE SEXO='M'),
3         (SELECT COUNT(*) AS "F" FROM FUNCIONARIO WHERE SEXO='F');
```

Fonte: O autor

	sexo	quantidade
1	M	2
2	F	3

Fonte: O autor.

📷 Número de funcionários por sexo: informações exibidas em uma única linha.

CONSULTA 02 + RESULTADO

Retornar a média salarial por sexo.

Fonte: O autor

Fonte: O autor

📷 Resultado consulta 02.

O SGBD realiza o agrupamento de dados de acordo com os valores da coluna SEXO. Em seguida, para cada grupo encontrado, a função AVG (SALARIO) é executada; e o resultado, exibido.

CONSULTA 03 + RESULTADO

Retornar, por mês de aniversário, a quantidade de colaboradores, o menor salário, o maior salário e o salário médio. Ordene os resultados por mês de aniversário.

Fonte: O autor

Fonte: O autor

📷 Resultado consulta 03.

O SGBD realiza o agrupamento de dados de acordo com o mês de nascimento dos funcionários. Depois, para cada grupo encontrado, as funções de agregação são executadas e, em seguida, exibidos os resultados. Perceba também que, na linha 4, utilizamos a função ROUND com objetivo de mostrar ao usuário final somente a parte inteira dos valores resultantes da média salarial.

CONSULTA 04 + RESULTADO

Retornar, por mês de aniversário, o mês, o sexo e a quantidade de colaboradores. Apresentar os resultados ordenados pelo mês.

Fonte: O autor

Fonte: O autor

📷 Resultado consulta 04.

O SGBD realiza o agrupamento de dados de acordo com os valores do mês de aniversário. Em seguida, no contexto de cada mês encontrado, mais um grupo é construído por sexo. Finalmente, para cada ocorrência mês/sexo, o número de colaboradores é calculado.

GRUPO DE DADOS COM GROUP BY E HAVING

Até o momento, utilizamos a cláusula WHERE para programar filtros em consultas, com condições simples ou compostas envolvendo colunas da tabela ou funções de data.

Contudo, você vai vivenciar situações onde será necessário estabelecer algum tipo de filtro, tendo como base um cálculo originado a partir de uma função de agregação, não sendo possível usar a cláusula WHERE. Nesses casos, utilizamos a cláusula HAVING, que serve justamente para esse propósito.

Vamos ver a seguir um exemplo de quando utilizar essa cláusula.

CONSULTA 05 + RESULTADO

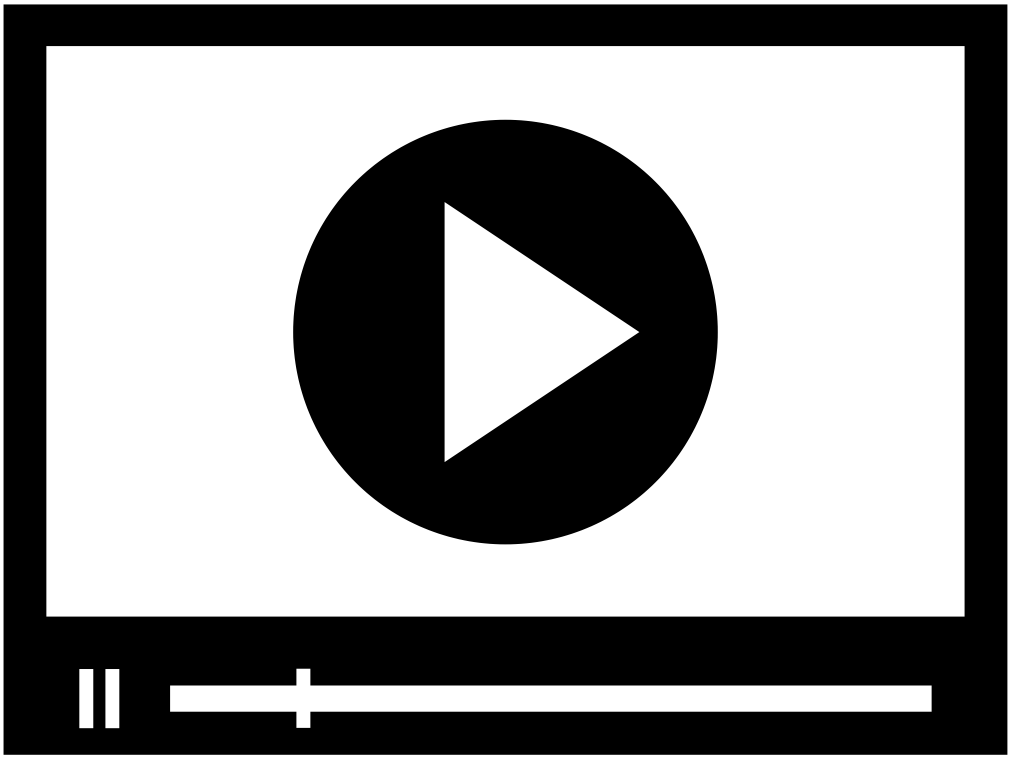
Suponha que o departamento de recursos humanos esteja estudando a viabilidade de oferecer bônus de 5% aos funcionários por mês de nascimento, mas limitado somente aos casos onde há mais de um colaborador aniversariando. Assim, para cada mês em questão, deseja-se listar o mês, o número de colaboradores e o valor do bônus.

Solução:

Fonte: O autor

Fonte: O autor
Resultado consulta 05.

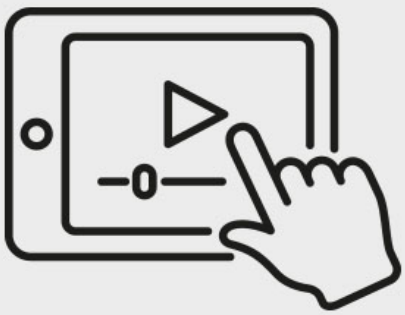
Note que estamos diante de uma estrutura de consulta muito similar ao código da consulta 03. Porém, estamos interessados em retornar somente o(s) registro(s) cujo valor da coluna **quantidade** seja maior que a unidade. Acontece que **quantidade** é uma coluna calculada com auxílio de uma função de agregação, não sendo possível programar um filtro na cláusula WHERE (WHERE QUANTIDADE>1). Assim, declaramos o filtro de interesse fazendo uso da cláusula HAVING, conforme linha 6 da consulta.



CONSULTAS COM O COMANDO SELECT E AS CLÁUSULAS GROUP BY E HAVING

Veja no vídeo a seguir como realizar **Consultas com o comando SELECT e as cláusulas GROUP BY e HAVING**

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Ao longo da nossa jornada, estudamos o projeto de consultas com o uso de agrupamento de dados. Percebemos que esse recurso é imprescindível quando temos interesse na extração de informações de caráter mais analítico a partir de alguma tabela, fazendo uso de funções de agregação associadas a uma ou diversas colunas.

Ainda, percebemos que, às vezes, a natureza do problema que estamos resolvendo requer o uso de filtro tendo como base o uso de alguma função de agregação. Para isso, fizemos uso da cláusula HAVING.

Agora é com você! Vamos realizar as atividades a seguir?

VERIFICANDO O APRENDIZADO

1. SUPONHA QUE EXISTA EM UM BANCO DE DADOS UMA TABELA DENOMINADA CLIENTE, ASSIM ESTRUTURADA: CLIENTE (CODIGOCLIENTE, NOME, SEXO, BAIRRO, RENDA). VOCÊ FOI SOLICITADO A ESCREVER UM COMANDO SQL PARA OBTER A RENDA MÉDIA DOS CLIENTES POR BAIRRO.
O COMANDO CORRETO É:

A) SELECT BAIRRO, MIN(RENDA)

FROM CLIENTE

GROUP BY BAIRRO

B) SELECT SEXO, SUM(RENDA)

FROM CLIENTE

GROUP BY BAIRRO

C) SELECT BAIRRO, AVG(RENDA)

FROM CLIENTE

GROUP BY BAIRRO

D) SELECT BAIRRO,MAX(RENDA)

FROM CLIENTE

GROUP BY SEXO

2. SUPONHA A EXISTÊNCIA DE UMA TABELA NO POSTGRESQL COM A SEGUINTE ESTRUTURA: PRODUTO (CODIGOP, NOME, ANO QUANTIDADE). SUPONHA TAMBÉM QUE A TABELA TENHA OS

SEGUINTES REGISTROS:

CODIGO P	NOME	ANO	QUANTIDADE
1	VIRTUS	2020	3
2	FIESTA	2014	1
3	CRUZE	2020	4
4	CAMARO	2018	1
5	KOMBI	1996	4
6	FOCUS	2016	3

ATENÇÃO! PARA VISUALIZAÇÃOCOMPLETA DA TABELA UTILIZE A ROLAGEM HORIZONTAL

QUAL CONSULTA A SEGUIR RETORNA MAIS DE DOIS RESULTADOS?

A) SELECT ANO,SUM(QUANTIDADE) AS TOTAL
FROM PRODUTO
GROUP BY ANO
HAVING SUM(QUANTIDADE)>1;

B) SELECT ANO,SUM(QUANTIDADE) AS TOTAL
FROM PRODUTO
GROUP BY ANO;

C) SELECT SUM(QUANTIDADE) AS TOTAL
FROM PRODUTO;

D) SELECT ANO, COUNT(*) AS TOTAL

FROM PRODUTO

WHERE QUANTIDADE>5

GROUP BY ANO;

GABARITO

1. Suponha que exista em um banco de dados uma tabela denominada CLIENTE, assim estruturada: CLIENTE (CODIGOCLIENTE, NOME, SEXO, BAIRRO, RENDA). Você foi solicitado a escrever um comando SQL para obter a renda média dos clientes por bairro. O comando correto é:

A alternativa **"C "** está correta.

Para recuperar corretamente os registros de interesse, é necessário agrupar os dados pela coluna BAIRRO e em seguida usar a função de média (AVG), tendo como base a coluna RENDA.

2. Suponha a existência de uma tabela no PostgreSQL com a seguinte estrutura: PRODUTO (CODIGOP, NOME, ANO QUANTIDADE). Suponha também que a tabela tenha os seguintes registros:

CODIGO P	NOME	ANO	QUANTIDADE
1	VIRTUS	2020	3
2	FIESTA	2014	1
3	CRUZE	2020	4
4	CAMARO	2018	1
5	KOMBI	1996	4

6	FOCUS	2016	3

Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

Qual consulta a seguir retorna mais de dois resultados?

A alternativa **"A "** está correta.

Para recuperar os registros de interesse, a consulta em questão retorna o total de automóveis por ano, no entanto, levando em conta somente os grupos em que o total seja maior que 1. Na prática, os anos 2014 e 2018 não farão parte dos resultados da consulta e os demais o farão, totalizando três resultados.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Neste tema, tratamos do comando SELECT da SQL no PostgreSQL. Vimos a sua sintaxe básica para consulta a uma tabela, no formato SELECT ... FROM ... WHERE.

Reconhecemos que, na cláusula SELECT, são especificadas as colunas da tabela a serem selecionadas, o que corresponde à operação de projeção da Álgebra Relacional.

Aprendemos que é possível especificar expressões e funções nesta cláusula. No caso específico do PostgreSQL, vimos que a execução de funções pré-definidas é realizada especificando o nome e os parâmetros da função na cláusula SELECT, omitindo as demais cláusulas do comando, inclusive a cláusula FROM. Em seguida, estudamos o uso da cláusula WHERE, que especifica a condição de seleção de linhas da tabela, o que corresponde à operação de restrição ou seleção da Álgebra Relacional. Por fim, aplicamos cláusulas adicionais do comando SELECT, como ORDER BY, GROUP BY e HAVING, todas implementadas no PostgreSQL em compatibilidade com o padrão da linguagem SQL.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.

AVALIAÇÃO DO TEMA

REFERÊNCIAS

AWS. **Tarefas comuns do administrador de banco de dados para PostgreSQL.** *In*: AWS. Consultado em meio eletrônico em: 30 mai. 2020.

BILECKI, L. F.; KALEMPA, V. C. **EasyRA**: Uma ferramenta para tradução de consultas em álgebra relacional para SQL. In: Computer On The Beach, 2015, Florianópolis. Computer on the Beach, 2015. p. 21-30.

CAMPOS, N. S. **Notas de Aula sobre Banco de Dados da professora Nathielly Campos.**

Disponível sob licença Creative Commons BR Atribuição – CC BY, 2020.

ELMASRI, R.; NAVATHE, S. Sistemas de Banco de Dados. 7. ed. São Paulo: Pearson, 2019.

POSTGRESQL. **Chapter 9. Functions and Operators.** *In*: PostgreSQL. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. **PostgreSQL Downloads.** *In*: PostgreSQL. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. **SELECT**. *In*: PostgreSQL. Consultado em meio eletrônico em: 30 mai. 2020.

EXPLORE+

Para aprofundar os seus conhecimentos sobre o assunto deste tema, leia:

> “Tarefas comuns do administrador de banco de dados para PostgreSQL”, um interessante material sobre tarefas do dia a dia de um administrador de banco de dados que atue com o PostreSQL. Você pode encontrá-lo no site da Amazon Web Services.

> BILECKI, L. F.; KALEMPA, V. C. **EasyRA: Uma ferramenta para tradução de consultas em álgebra relacional para SQL.** *In*: Computer on the Beach, 2015. É importante saber que parte considerável da linguagem SQL é baseada na teoria de Álgebra Relacional. Trata-se de uma álgebra que envolve diversos operadores sobre relações. Neste trabalho, você aprenderá sobre as operações básicas da Álgebra Relacional, e conhecerá uma ferramenta para praticar comandos em álgebra e visualizar o respectivo comando na linguagem SQL.

CONTEUDISTA

Nathielly de Souza Campos

🔗 CURRÍCULO LATTES